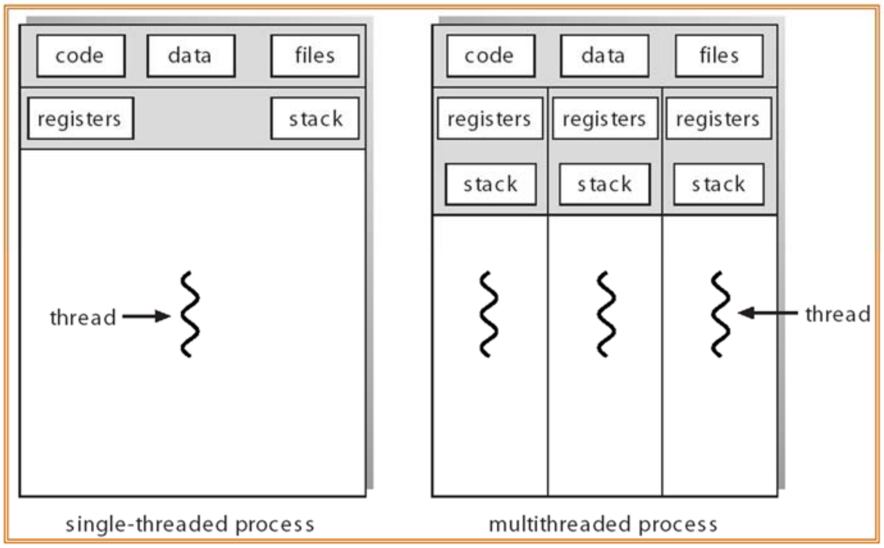
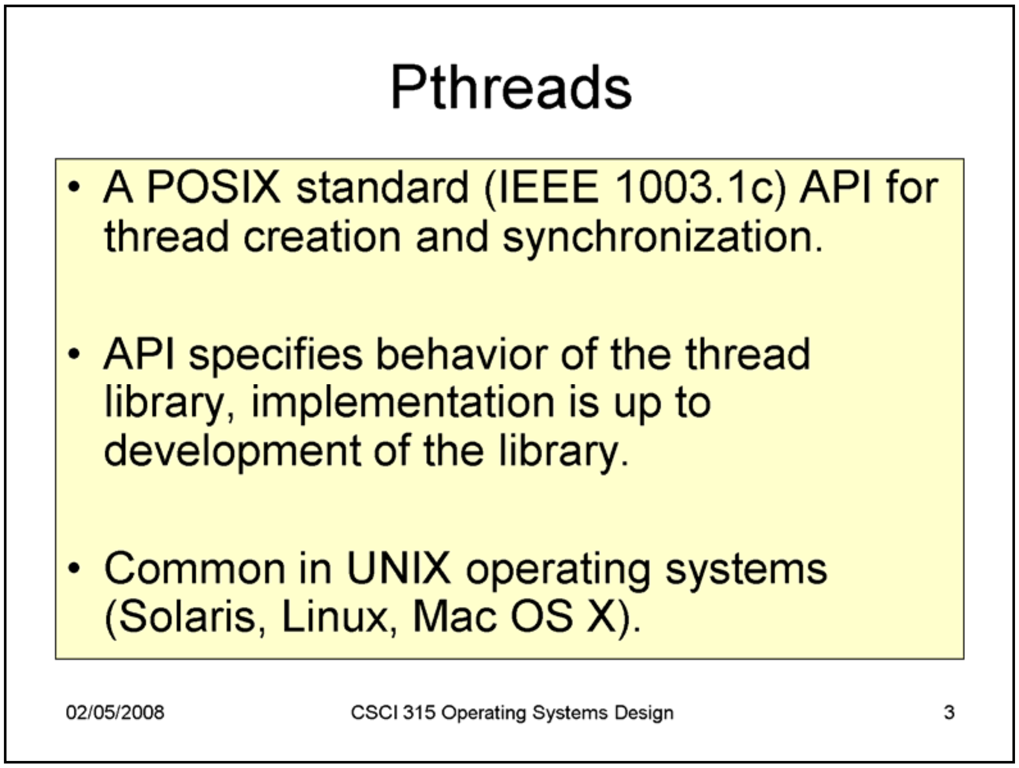
BUCKNELL UNIVERSITY
Computer Science
CSCI 315 Operating Systems Design

Java Threads

<u>Notice:</u> The slides for this lecture have been largely based on those accompanying an earlier version of the course text *Operating Systems Concepts with Java*, by Silberschatz, Galvin, and Gagne. Many, if not all, the illustrations contained in this presentation come from this source.

Multithreading

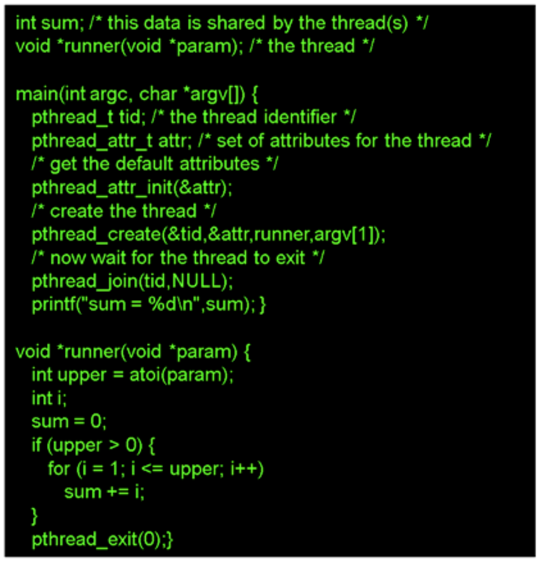




Pthreads

- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization.
- API specifies behavior of the thread library, implementation is up to development of the library.
- Common in UNIX operating systems (Solaris, Linux, Mac OS X).

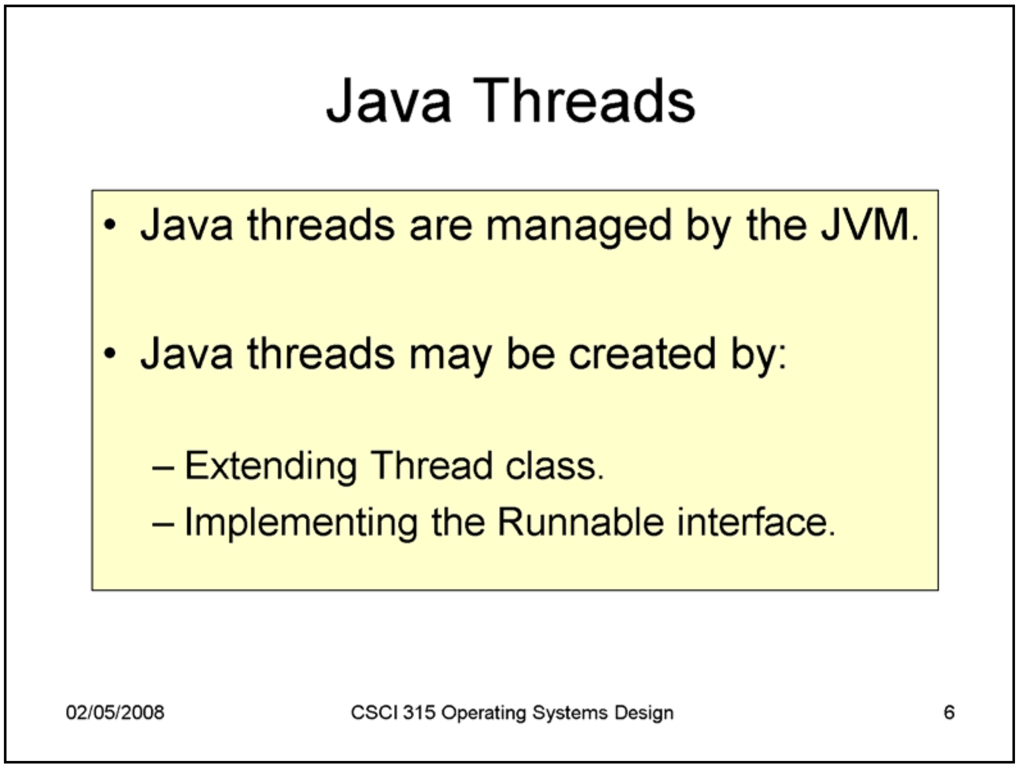
Pthreads



```
int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* the thread */

main(int argc, char *argv[]) {
  pthread_t tid; /* the thread identifier */
  pthread_attr_t attr; /* set of attributes for the thread */
  /* get the default attributes */
  pthread_attr_init(&attr);
  /* create the thread */
  pthread_create(&tid,&attr,runner,argv[1]);
  /* now wait for the thread to exit */
  pthread_join(tid,NULL);
  printf("sum = %d\n",sum); }

void *runner(void *param) {
  int upper = atoi(param);
  int i;
  sum = 0;
  if (upper > 0) {
    for (i = 1; i <= upper; i++)
      sum += i;
  }
  pthread_exit(0);}
```

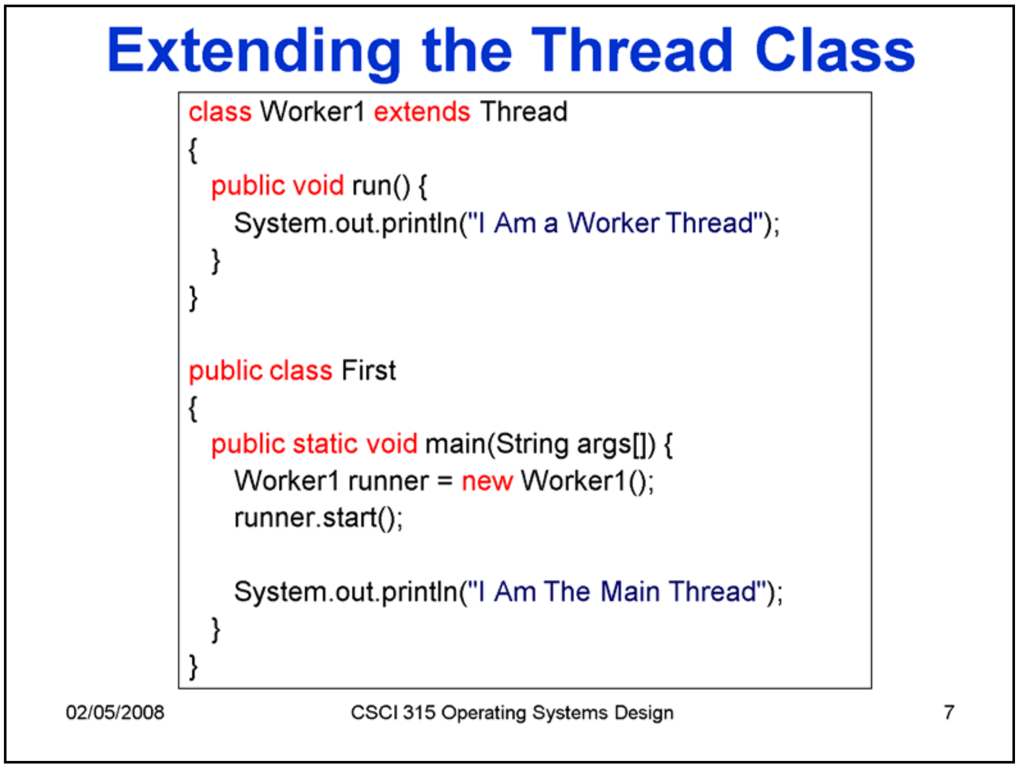
Linux Threads

- Linux refers to them as *tasks* rather than *threads.*
- Thread creation is done through **clone()** system call.
- **clone()** allows a child task to share the address space of the parent task (process).



Java Threads

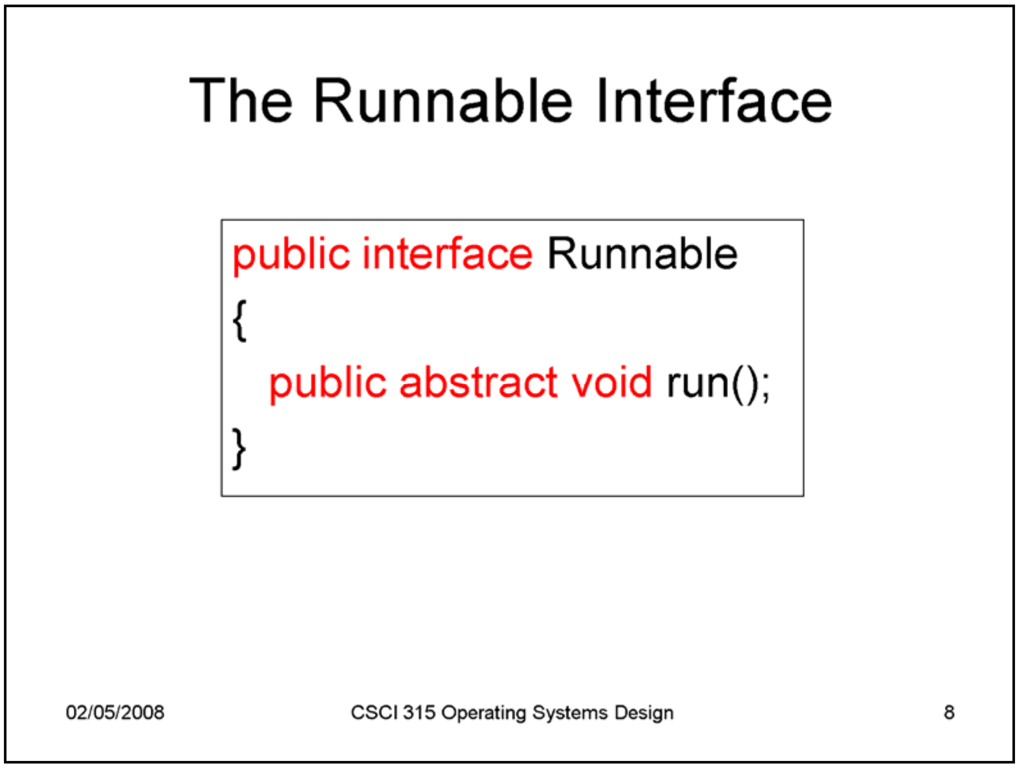
- Java threads are managed by the JVM.
- Java threads may be created by:
  - Extending Thread class.
  - Implementing the Runnable interface.



Extending the Thread Class

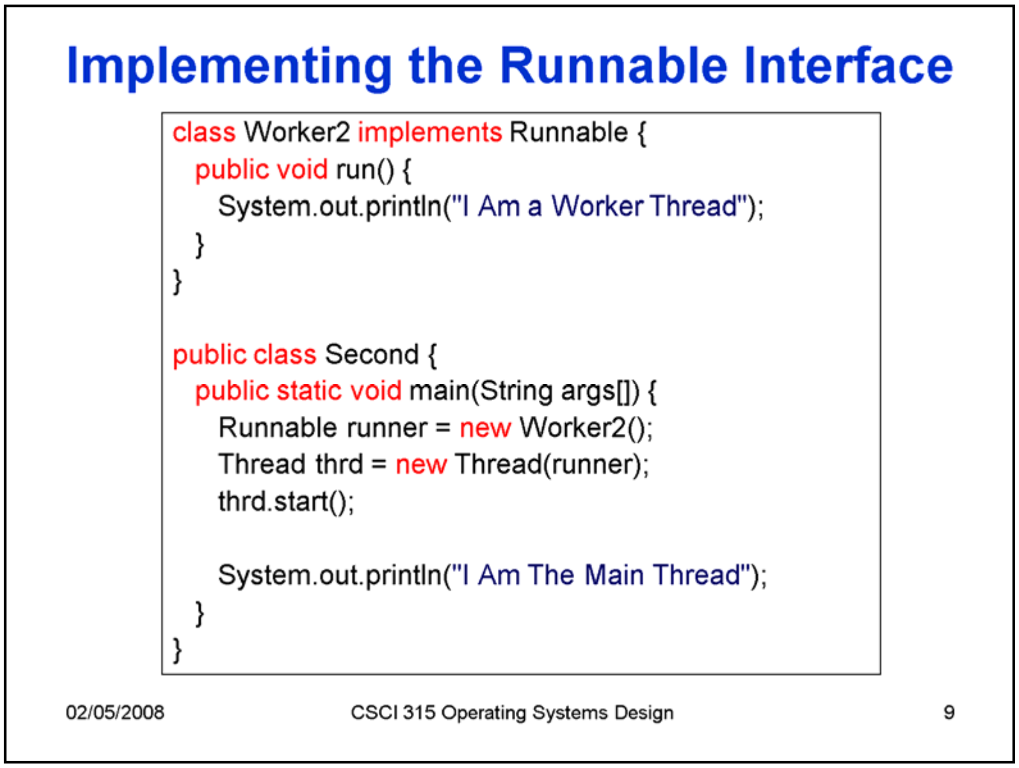
```
class Worker1 extends Thread
{
   public void run() {
      System.out.println("I Am a Worker Thread");
   }
}

public class First
{
   public static void main(String args[]) {
      Worker1 runner = new Worker1();
      runner.start();

      System.out.println("I Am The Main Thread");
   }
}
```

02/05/2008 CSCI 315 Operating Systems Design 7



The Runnable Interface

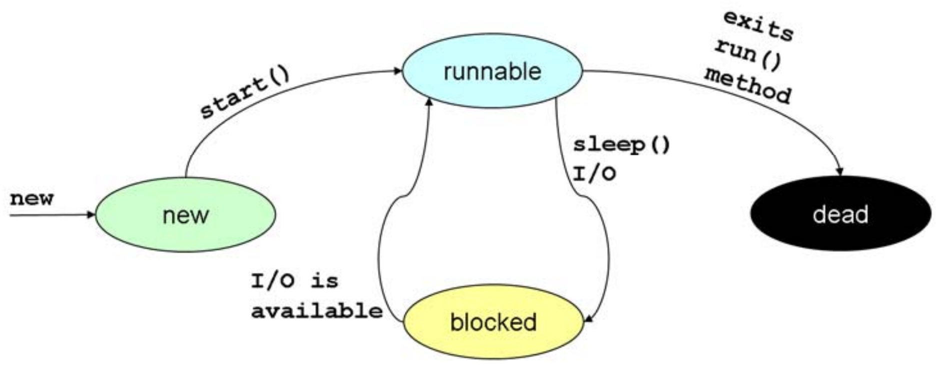
```
public interface Runnable
{
   public abstract void run();
}
```



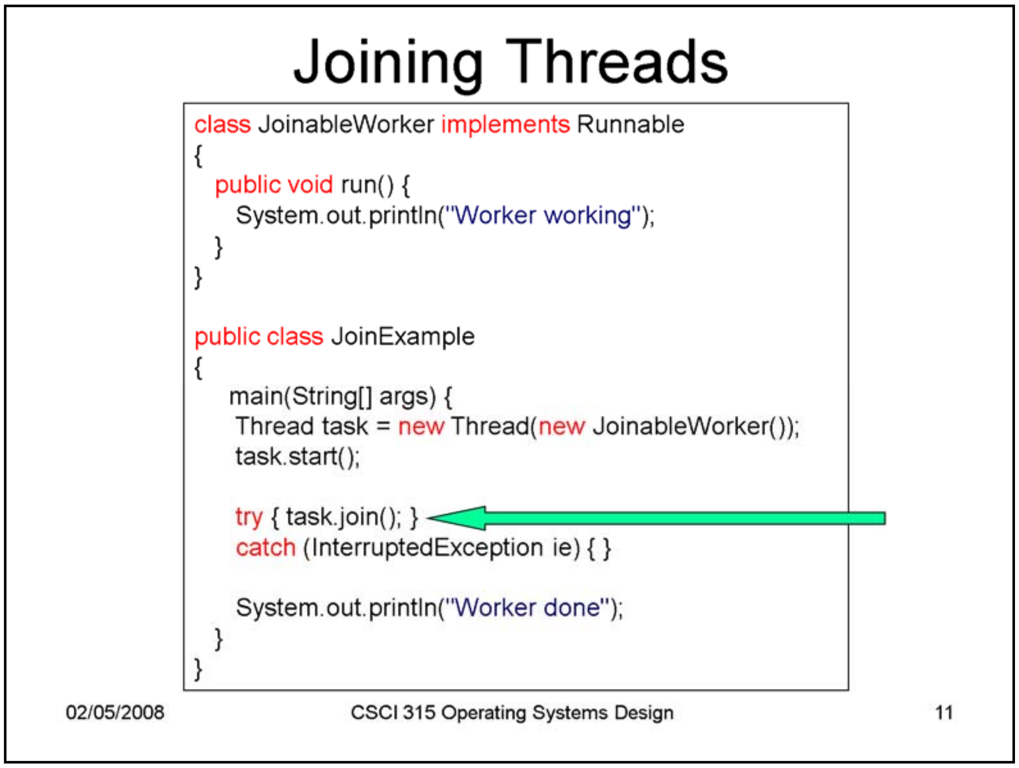
Implementing the Runnable Interface

```
class Worker2 implements Runnable {
  public void run() {
    System.out.println("I Am a Worker Thread");
  }
}

public class Second {
  public static void main(String args[]) {
    Runnable runner = new Worker2();
    Thread thrd = new Thread(runner);
    thrd.start();

    System.out.println("I Am The Main Thread");
  }
}
```

Java Thread States

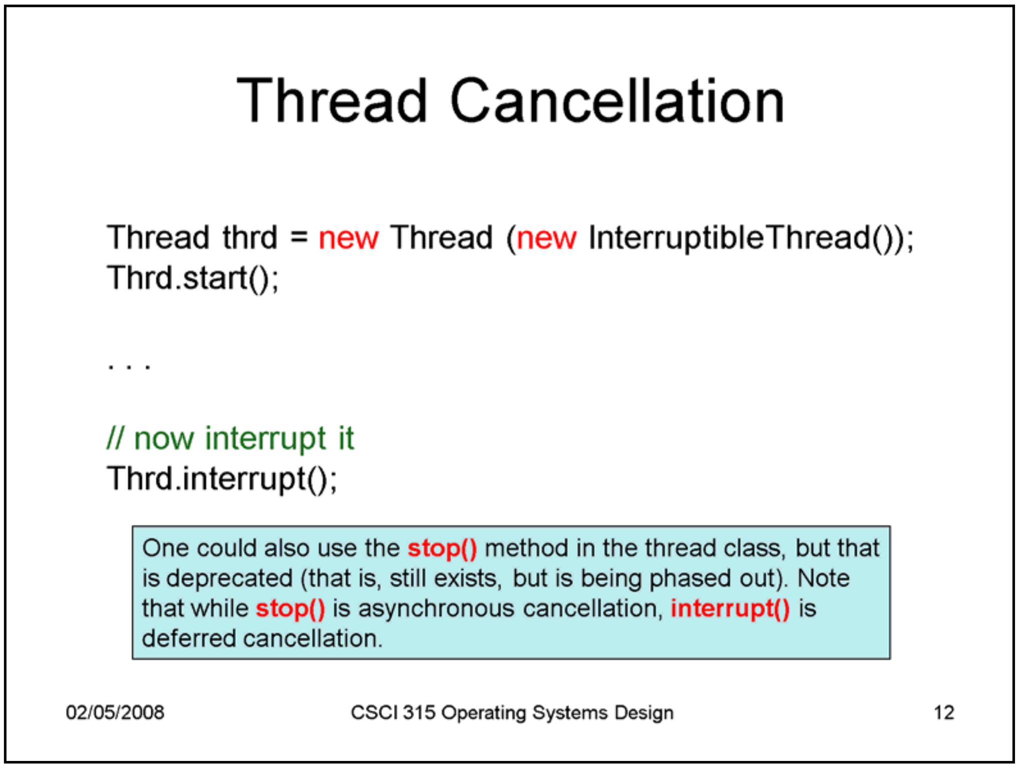


02/05/2008 CSCI 315 Operating Systems Design 10



Joining Threads

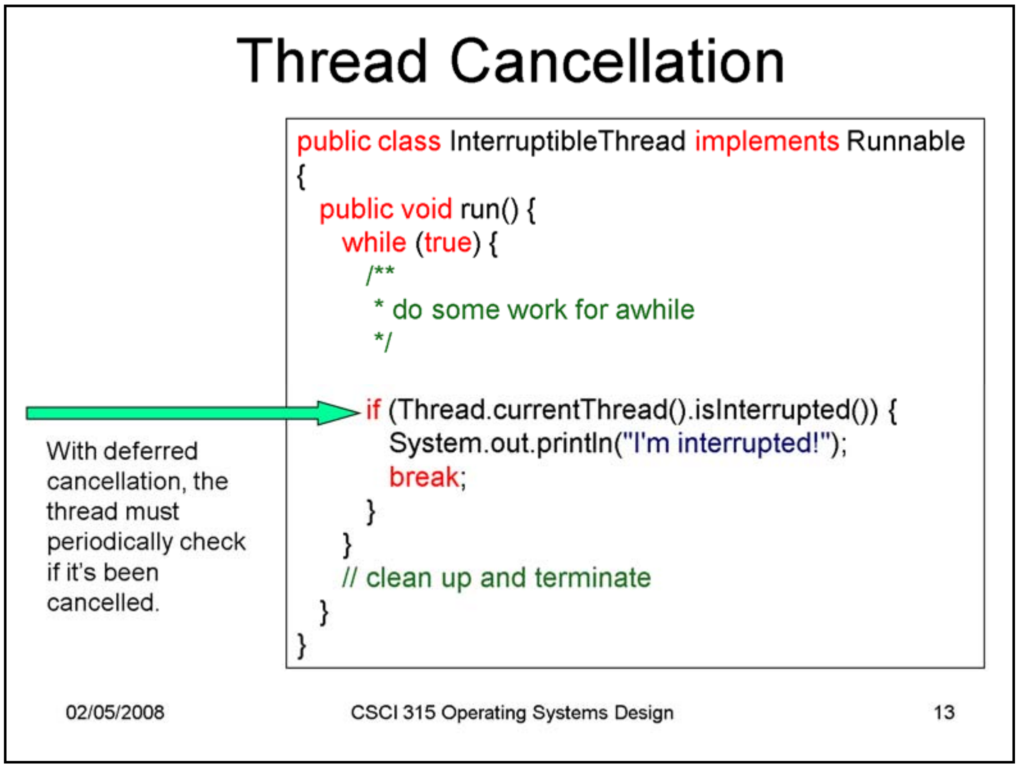
```
class JoinableWorker implements Runnable
{
  public void run() {
    System.out.println("Worker working");
  }
}

public class JoinExample
{
    main(String[] args) {
     Thread task = new Thread(new JoinableWorker());
     task.start();

     try { task.join(); }
     catch (InterruptedException ie) { }

     System.out.println("Worker done");
  }
}
```



Thread Cancellation

```
Thread thrd = new Thread (new InterruptibleThread());
Thrd.start();

. . .

// now interrupt it
Thrd.interrupt();
```

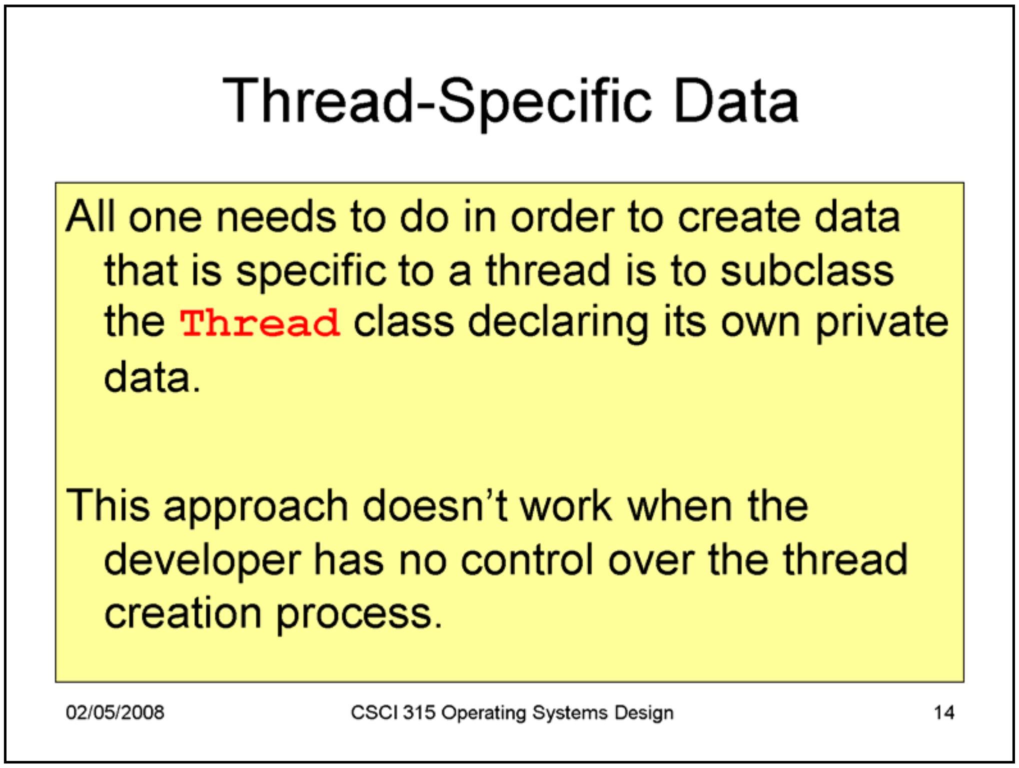
One could also use the **stop()** method in the thread class, but that is deprecated (that is, still exists, but is being phased out). Note that while **stop()** is asynchronous cancellation, **interrupt()** is deferred cancellation.



Thread Cancellation

```
public class InterruptibleThread implements Runnable
{
   public void run() {
      while (true) {
         /**
          * do some work for awhile
          */

         if (Thread.currentThread().isInterrupted()) {
            System.out.println("I'm interrupted!");
            break;
         }
      }
      // clean up and terminate
   }
}
```

With deferred cancellation, the thread must periodically check if it's been cancelled.

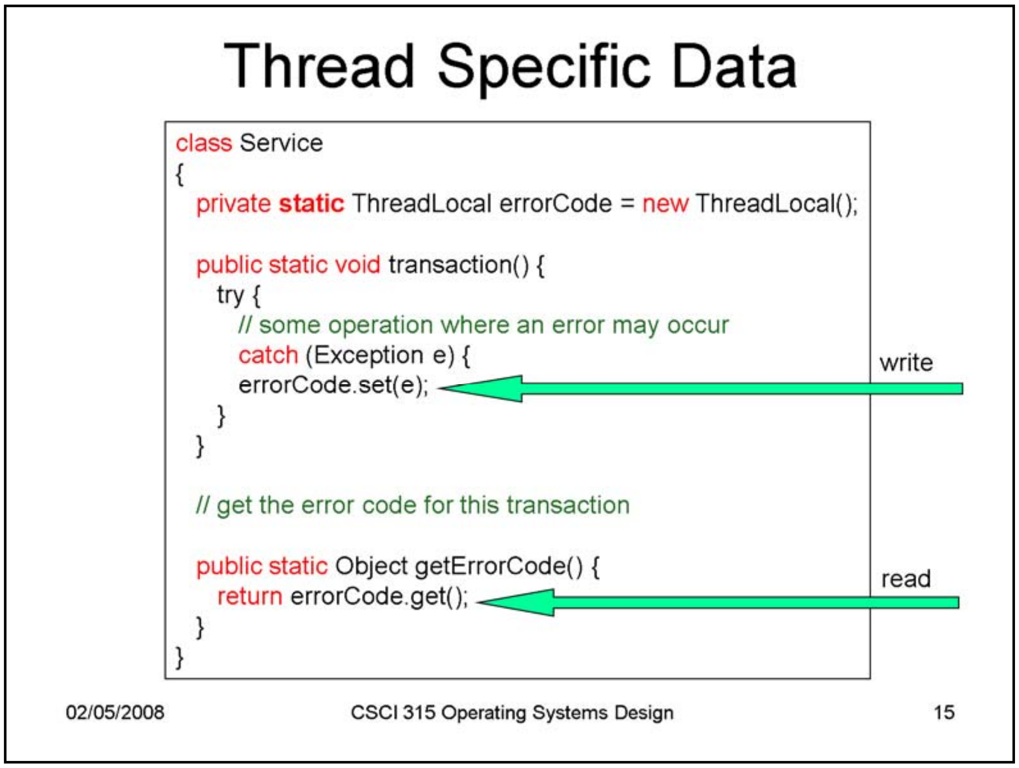


Thread-Specific Data

All one needs to do in order to create data that is specific to a thread is to subclass the `Thread` class declaring its own private data.

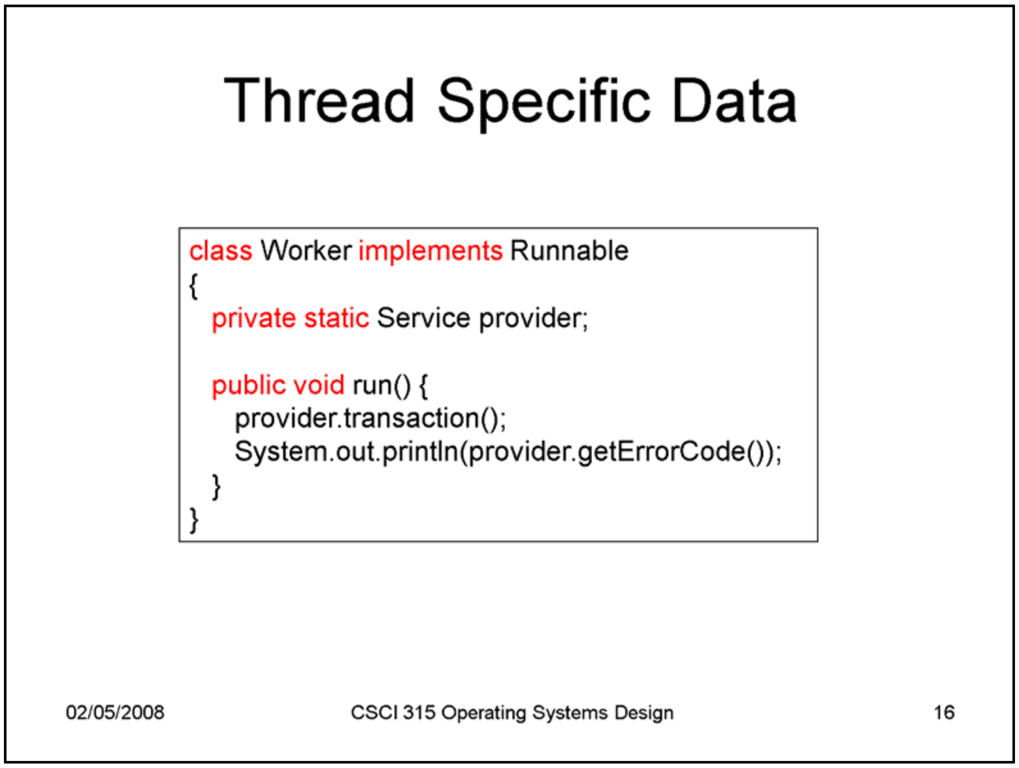
This approach doesn't work when the developer has no control over the thread creation process.

02/05/2008 CSCI 315 Operating Systems Design 14



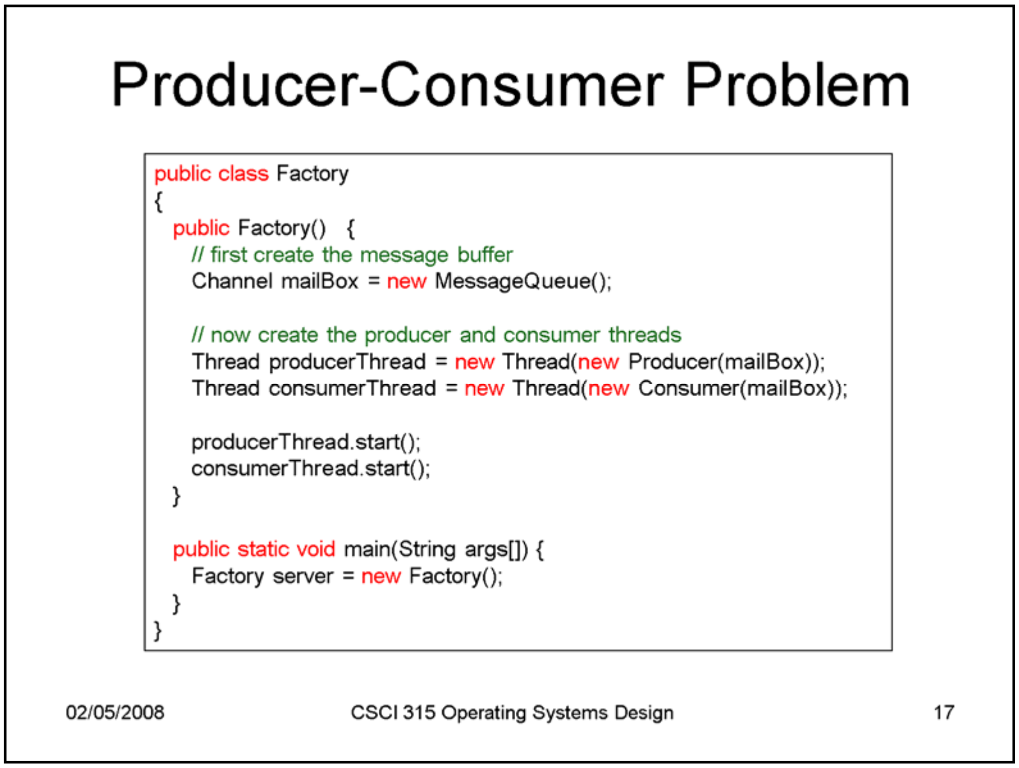
Thread Specific Data

```
class Service
{
   private static ThreadLocal errorCode = new ThreadLocal();

   public static void transaction() {
      try {
         // some operation where an error may occur
         catch (Exception e) {
         errorCode.set(e);
      }
   }

   // get the error code for this transaction

   public static Object getErrorCode() {
      return errorCode.get();
   }
}
```



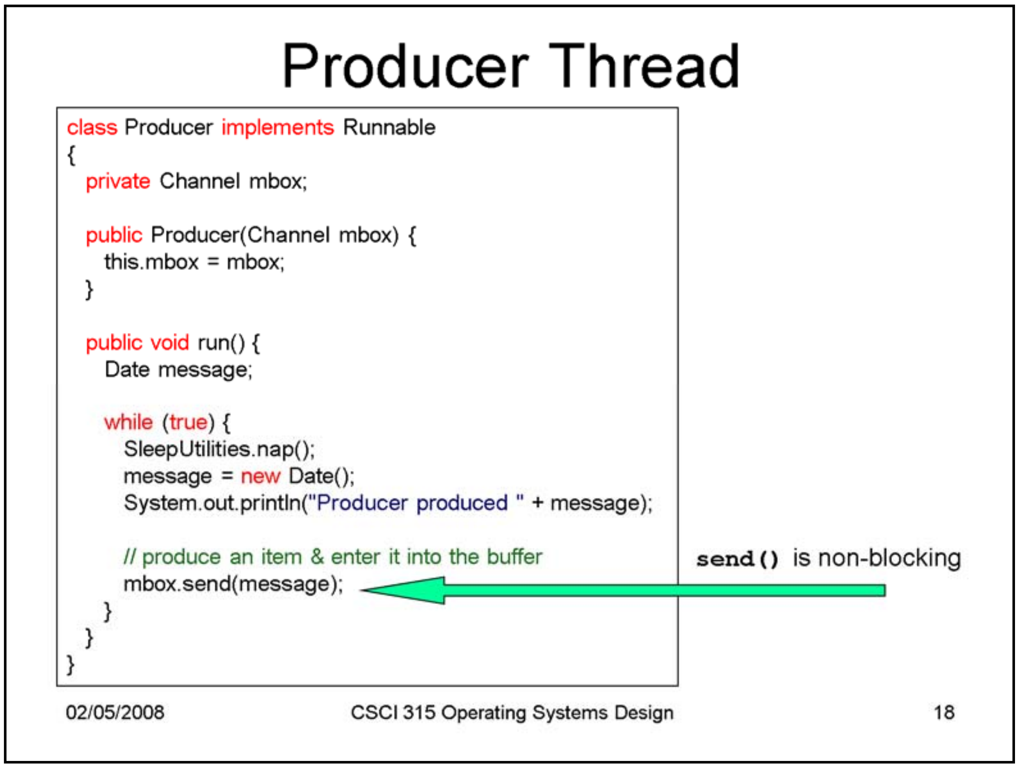
Thread Specific Data

```
class Worker implements Runnable
{
  private static Service provider;

  public void run() {
    provider.transaction();
    System.out.println(provider.getErrorCode());
  }
}
```



Producer-Consumer Problem

```
public class Factory
{
  public Factory()  {
    // first create the message buffer
    Channel mailBox = new MessageQueue();

    // now create the producer and consumer threads
    Thread producerThread = new Thread(new Producer(mailBox));
    Thread consumerThread = new Thread(new Consumer(mailBox));

    producerThread.start();
    consumerThread.start();
  }

  public static void main(String args[]) {
    Factory server = new Factory();
  }
}
```

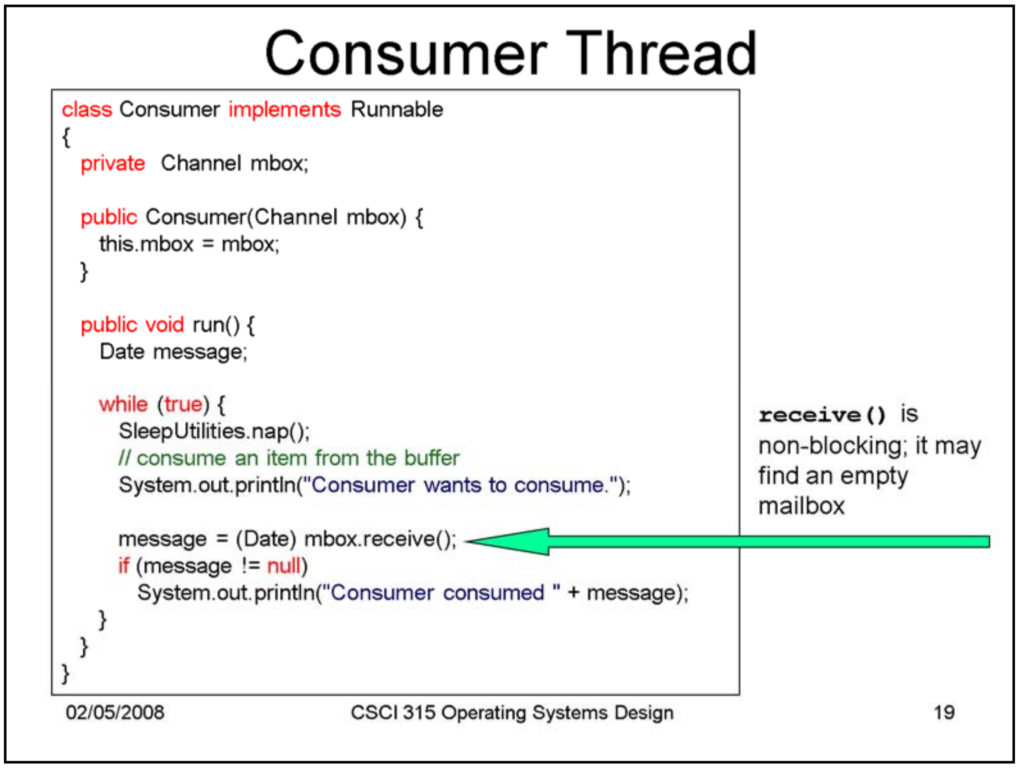


Producer Thread

```
class Producer implements Runnable
{
  private Channel mbox;

  public Producer(Channel mbox) {
    this.mbox = mbox;
  }

  public void run() {
    Date message;

    while (true) {
      SleepUtilities.nap();
      message = new Date();
      System.out.println("Producer produced " + message);

      // produce an item & enter it into the buffer
      mbox.send(message);
    }
  }
}
```

`send()` is non-blocking

02/05/2008 CSCI 315 Operating Systems Design 18



Consumer Thread

```
class Consumer implements Runnable
{
  private  Channel mbox;

  public Consumer(Channel mbox) {
    this.mbox = mbox;
  }

  public void run() {
    Date message;

    while (true) {
      SleepUtilities.nap();
      // consume an item from the buffer
      System.out.println("Consumer wants to consume.");

      message = (Date) mbox.receive();
      if (message != null)
        System.out.println("Consumer consumed " + message);
    }
  }
}
```

`receive()` is non-blocking; it may find an empty mailbox

02/05/2008 CSCI 315 Operating Systems Design 19